Čo-to o SCTimeri alebo elegantné riešenie ako pomocou SCTimera a DMA prenosu generovať sínusovku

Čo-to o NXP LPC SCTimeri alebo elegantné riešenie ako pomocou SCTimera a DMA prenosu generovať sínusovku

Niektoré ARM Procesory LPC od fy.NXP, obsahujú periférium s názvom SCTimer - State Configurable Timer.

Náznaky o fungovaní časovača sa objavili už v komentároch pri predstavení rodiny LPC8xx no nič konkrétne.

Skúsim len letmo - viacej sa dá nájsť na stránke NXP venovanej SCTimeru, alebo v kuchárke a samozrejme aj v manuále príslušného mikrokontroléra.

SCTimer je kombinácia klasického časovača s programovateľnou logikou.

Táto logika zavádza nový pojem "Event" (udalosť).
Event je vlastne stavebným prvkom SCTimera. Dá sa naň pozerať ako na jeden bit, ktorý priamo dokáže ovplyvňovať chovanie nielen SCTimera samotného, ale fyzicky dotiahnutý aj k ďalším perifériám MCU.
Event môže nastaviť dokonca aj IO pin a naopak zmena na IO pine môže vygenerovať event.
Event môže nastavovať prerušenie NVIC, môže spúšťať DMA prenos, samozrejme ovplyvňovať samotný časovač - nulovať, pozastaviť, zmeniť smer počítania.

Ďalším pojmom je "State" (stav), ktorý môže nadobúdať hodnoty 0 až n , pričom n je závislé od konkrétneho MCU. Dnes sa pohybuje od 2 do 32.
Tento stav ďalej komplikuje použiteľnosť SCTimera, pretože násobí použitie eventov - stav je menený od eventov, ale naopak event môže nastať len v konkrétnych stavoch (alebo pri všetkých)

Tým, že je pomocou konfiguračných registrov "hardwérovo" prepojiteľný ku svojmu okoliu, nepotrebuje pri práci spoluprácu MCU jadra. Pracuje nezávisle.
Kombinácia jeho vlastností robí z tejto periférie veľmi silný nástroj.

Príklad.
Pri generovaní obdĺžnikového signálu sú potrebné dva eventy - jeden event nastavuje Log.1 na IO pine, druhý zasa nastavuje Log.0.
SCTimer je skonfigurovaný tak, že pri dopočítaní k žiadanej hodnote generuje Event 1 (ktorému sme povedali čo má robiť), a pri dosiahnutí maxima (pretečení) generuje Event 0 s možnosťou automaticky znulovať časovač a znovu spustiť počítanie.

Pomocou týchto eventov je teda možné okamžite spraviť to, čo pri bežných časovačoch musí vykonať obslužná rutina.

Možno aj z príkladu je teda zrejmé, že obmedzenie pre SCTimer je hlavne počet Eventov, ktorými disponuje.

Pre mňa zaujímavé je HW prepojenie na ďalšie periférie MCU.
V jednej aplikácii som generoval sínusový signál tvorený PWM (a nejaký ten hardwér v podobe RC článku) s klasickým časovačom, no réžia jadra bola príliš veľká. Jadro sa veľmi silno venovalo len obsluhe časovača - reload zo sínusovej tabuľky (predpočítaná).

Takže po viacerých pokusoch, sa mi podarilo prepojiť SCTimer s DMA kanálmi, čo malo za následok autonómne generovanie sínusového signálu zo sínusovej tabuľky kompletne bez zásahu jadra MCU.
To má prácu len pri zmene frekvencie signálu, kedy znovu prepočíta hodnoty pre časovač a uloží ich do sínusovej tabuľky.
Následne som test rozšíril až na tri PWM výstupy (generovanie 3 sínusových signálov) s možnosťou nastaviť fázový posun medzi nimi - opäť fungujúce autonómne.
Len jadro dostáva prácu už aj pri zmene fázového posunu - zmenou adries SRC pre DMA prenos....

Pre jeden signál s nastaviteľnou frekvenciou je konfigurácia jednoduchá.
DMA prenos je skonfigurovaný s pomocou jedného descriptora, kde SRC ukazuje na koniec sin. tabuľky, a počet prenesených hodnôt je veľkost DMA tabuľky.

Takže samotný DMA transfer nastavuje Transfer descriptor s príznakom RELOAD (ano, po použití ho znova použi), SRCINC=1 (inkrementuj zdrojovú adresu, DSTINCR=0 (destination nemeň)XFERCOUNT je počet hodnôt v sin. tabuľke.

```
Chip_DMA_Table[DMA_SCTRELOAD_0].xfercfg             =
                        DMA_XFERCFG_CFGVALID                // Channel descriptor is considered valid
                      | DMA_XFERCFG_RELOAD
                      | DMA_XFERCFG_WIDTH_32                // 8,16,32 bits allowed
                      | DMA_XFERCFG_SRCINC_1                // increment src address
                      | DMA_XFERCFG_DSTINC_0                // constant dest address
                      | DMA_XFERCFG_XFERCOUNT(SIN_STEPS);
Chip_DMA_Table[DMA_SCTRELOAD_0].source          = ((uint32_t) ( &SinTable[SIN_STEPS]));                 // set source
Chip_DMA_Table[DMA_SCTRELOAD_0].dest            = ((uint32_t)(&LPC_SCT->MATCHREL[1].U));                // destination = address of SCTimer -> reload register
Chip_DMA_Table[DMA_SCTRELOAD_0].next            = ((uint32_t)(&Chip_DMA_Table[DMA_SCTRELOAD_0]));  // next descriptor = still same one
```

Trigger je hardwérový (zo vstupného PINMUX-u), prenos je typu burst s burstpower = 1 (prenes všetko ale po jednom) - trigger edge (trigger aktivuj hranou):

```

LPC_DMA->DMACH[DMA_SCTRELOAD_0].CFG =
                          DMA_CFG_HWTRIGEN
                        |       DMA_CFG_TRIGTYPE_EDGE        // edge sensitive
                        |       DMA_CFG_TRIGPOL_HIGH         // rising edge

```

```
                    |       DMA_CFG_TRIGBURST_BURST      // burst transfer
7.
                    |       DMA_CFG_BURSTPOWER_1         // one transfer
8.
                    |       DMA_CFG_DSTBURSTWRAP         // dst address without change
9.
                    |       DMA_CFG_CHPRIORITY(3);                  // - with priority 3/7
```

Týmto mám zaručený prenos pri každom triggeri postupne jednej reload hodnoty zo sin. tabuľky do reload registru timera.

Ešte nastaviť trigger na SCTimer event:

```
1.
    LPC_DMATRIGMUX->DMA_ITRIG_INMUX[DMA_SCTRELOAD_0] = (uint32_t) DMATRIG_SCT0_DMA1;      // spustanie od SCT_DMA1            - používam DMA kanál 1
```

No a samozrejme nezáživné omáčky:

```
1.
    LPC_DMA->CTRL = 0;                                                                                  // Disable DMA controller
2.
    LPC_SYSCTL->SYSAHBCLKCTRL = (1 << SYSCTL_CLOCK_DMA) | (LPC_SYSCTL->SYSAHBCLKCTRL &amp;amp; ~SYSCTL_SYSAHBCLKCTRL_RESERVED); // DMA initialization -          enable DMA clocking and reset DMA if needed
3.
    LPC_DMA->SRAMBASE = ((uint32_t) (Chip_DMA_Table));
```

DMA povolím a nastavím samotný SCTimer.

```
1.
    LPC_DMA->DMACH[DMA_SCTRELOAD_0].XFERCFG = Chip_DMA_Table[DMA_SCTRELOAD_0].xfercfg;                  // nastavenie descriptora v HW registri
2.
```

```
LPC_DMA->DMACOMMON[0].SETVALID = (1 << DMA_SCTRELOAD_0);                    // nasavenie validity DMA
```

3.
```
LPC_DMA->CTRL = 1;                                                          // a povolenie DMA kontroléra
```

No a nastavenie SCTimera

1.
```
LPC_SCT->CONFIG = SCT_CONFIG_32BIT_COUNTER | SCT_CONFIG_CLKMODE_BUSCLK | SCT_CONFIG_AUTOLIMIT_L | SCT_CONFIG_AUTOLIMIT_H;       // jedne timer, BUS Clocked, auto - reload pri              MATCHREL[0]
```

2.

3.
```
LPC_SCT->REGMODE_U = 0x0000;                                                // 0: pracujeme ako MATCH, nie ako CAPTURE register
```

4.

5.
```
LPC_SCT->CTRL_U = SCT_CTRL_HALT_L |             SCT_CTRL_HALT_H | SCT_CTRL_CLRCTR_L | SCT_CTRL_CLRCTR_H;  // zastav počítadlo - malo by sa zastaviť pri každej konfigurácii
```

6.
```
LPC_SCT->CTRL_U |=      SCT_CTRL_BIDIR_L(0) |                               // počítaj smerom hore
```

7.
```
SCT_CTRL_PRE_L(0);                                                          // set prescale to n+1
```

Init SCTimera a pripojenie GPIO na OUT_0:

1.
```
Chip_Clock_EnablePeriphClock(SYSCTL_CLOCK_SWM);                             // Enable SWM clock before altering SWM
```

2.
```
Chip_SWM_MovablePinAssign(SWM_SCT_OUT0_O, GPIO.Pin);                        // Connect SCT output 0 to MCU              Pin
```

3.

```
    Chip_Clock_DisablePeriphClock(SYSCTL_CLOCK_SWM);

4.
    Chip_SCT_Init(LPC_SCT);                                                    // povol STC periferiu,
nastav CLOCK a daj RESET

5.
```

A nasavenie Eventov samotných:

```
1.
// EVENT  0 - zakladna cela perioda PWM - ukoncenie/start cyklu - nastavuje vystup na log. 1

2.
    LPC_SCT->MATCHREL[EV_0].U = 0;                                  // znuluj match register -          reload by user


3.
    LPC_SCT->EV[EV_0].STATE =  0xFF;                                  // Event  moze nastat kedykolvek - pri
akomkoľvek STATE

4.
    LPC_SCT->EV[EV_0].CTRL  =  0x00 |                                 // Event - MATCHSEL:    0x00. Nastane pri
zhode s MATCH[0]

5.
                               (0 << 4) |                       // - HEVENT:    0. Vyvolava sa od L countera
(alebo unified ako v tomto prípade)

6.
                               (1 << 5) |                       // - OUTSEL:    1. Vybera vystup nastaveny v
IOSEL

7.
                               ((0x01 <<          (OUT_0 )) << 6) |    // - IOSEL:             0x01. nastav
vystup c.0 [0..5]

8.
                               (0x00 << 10) |             // - IOCOND:    0x00. nepouzite

9.
                               (0x01 << 12) |             // - COMBMODE: 0x01. Event nastava len pri MATCH
zhode

10.
                               (0 << 14) |                      // - STATELD:   pripocitaj/prepis hodnotu stavu.
1: prepisujeme state hodnotu

11.
                               (0x00 << 15) |             // - STATEV:    0x01. Novy STATE po tomto evente
```

```
12.                                         (0 << 20) |                 // - MATCHMEM: 0. Meni porovnavanie z = na >=.
13.                                         (0x01 << 21) |              // - DIRECTION:0x01. Event vznikne len pocas pocitania smerom hore.
14.                                         SCT_EVn_CTRL_RESERVED;      // - .....
15.     LPC_SCT->OUT[OUT_0].SET = (1 << EV_0);                          // Event 0 nastavuje fizicky Out 0 - tuto mením stav GPIO
16.     LPC_SCT->DMAREQ1 = 1 << EV_0 ;                                  // Aktivuj DMA1 prenos od match[0] - zabezpečí reload hodnoty
17.
18.   // EVENT  1 - D.C PWM signálu nastavuje výstup do log.0
19.     LPC_SCT->MATCHREL[EV_1].U =  1;                                 // znuluj match register -  reloaduje ho DMA
20.     LPC_SCT->EV[EV_1].STATE =  0xff;                                // Event  moze  nastat kedykolvek
21.     LPC_SCT->EV[EV_1].CTRL  =  0x01 |                               // Event -  MATCHSEL:  0x01. Nastane pri zhode s MATCH[1]
22.                                         (0 << 4) |                  // - HEVENT:  0. Vyvolava sa od L countera
23.                                         (1 << 5) |                  // - OUTSEL:  1. Vybera vystup nastaveny v IOSEL
24.                                         ((0x01 <<  (OUT_0 )) << 6) |  // - IOSEL:  0x01. nastav vystup c.0 [0..5]
25.                                         (0x00 << 10) |              // - IOCOND:  0x00. nepouzite
26.                                         (0x01 << 12) |              // - COMBMODE: 0x01. Event nastava len pri MATCH zhode
27.                                         (0 << 14) |                 // - STATELD:  pripocitaj/prepis hodnotu stavu. 1: prepisujeme state hodnotu
28.
```

```
                                    (0x00 << 15) |              // - STATEV:    0x02. Novy STATE po tomto evente
29.
                                    (0 << 20) |                 // - MATCHMEM: 0. Meni porovnavanie z = na
    >=.
30.
                                    (0x01 << 21) |              // - DIRECTION:0x01. Event vznikne len pocas
    pocitania smerom hore.
31.
                                    SCT_EVn_CTRL_RESERVED;      // -
    ...........................................................
32.
        LPC_SCT->OUT[OUT_0].CLR |= (1 << EV_1);                                                    //
    Event 1 nuluje Out 0
```

A spusti celý cirkus:

```
1.
        LPC_SCT->CTRL_L &= ~SCT_CTRL_HALT_L;                    // start Low/unified counter
```

Týtmo je dosiahnuté generovanie sínusového signálu na pine GPIO.Pin, podľa tabuľky SinTable s frekvenciou nastavenou v LPC_SCT->MATCHREL[0].U

Tento príspevok má slúžiť len ako nasmerovanie pre podobné hračky. Nechcem (a nemôžem) zobraziť kompletný zdrojový kód, ale myslím že aj tieto fragmenty pomôžu pri pochopení funkcie SCTimeru....